63-3-2

FIFTH SEMI-ANNUAL REPORT
STUDY ON
APPLICATION OF CODING THEORY

1 July 1962 - 30 December 1962

Report No. PIBMRI-895.5-63
Contract No. DA-36-039 sc-78972

for

U. S. Army Signal Research and Development Labs.
Department of the Army Project No. 3A99-20-001-07

15 February 1963

MRI

POLYTECHNIC INSTITUTE OF BROOKLYN
MICROWAVE RESEARCH INSTITUTE
ELECTROPHYSICS DEPARTMENT

Microwave Research Institute
Polytechnic Institute of Brooklyn
55 Johnson Street
Brooklyn 1, New York

Report No. PIBMRI-895.5-63
Dept. of the Army Project No.
3A99-20-001-07

FIFTH SEMI-ANNUAL REPORT

STUDY ON APPLICATION OF CODING THEORY

1 July 1962 - 30 December 1962

PREPARED FOR

Department of the Army

USA Signal Research and Development Labs.

Fort Monmouth, New Jersey

Under

Contract No. DA-36-039 sc-78972

Title Page
Acknowledgment
Abstract
Table of Contents
33 Pages of Text
Distribution List

Submitted: Arthur E. Laemmel

A. E. Laemmel
Research Associate
Professor

Brooklyn 1, New York
15 February 1963

ACKNOWLEDGMENT

The work reported herein was sponsored by the Department of the Army, USA Signal Research and Development Laboratories, under Contract DA-36-039 sc-78972.

ABSTRACT

Several miscellaneous results are given in the theory of digital codes and finite state machines. Of particular interest in either case is the problem of ergodicity, i. e. whether a machine (which may be a decoder) can be reset to the correct state with the proper input sequence and with no knowledge of the present states or outputs. Some of the results given are algebraic methods and other tools useful in examining ergodicity and related problems. Consideration is also given to the synthesis of sequential machines to economically solve tasks which are essentially non-sequential.

TABLE OF CONTENTS

Purpose

The present contract has as its aim a study of certain properties of digital codes used for communication, both in general and as applied to Signal Corps porblems. Some consideration is also given to finite state machines since they are necessary for encoding and decoding operations, and since they are analyzed by essentially the same mathematical tools.

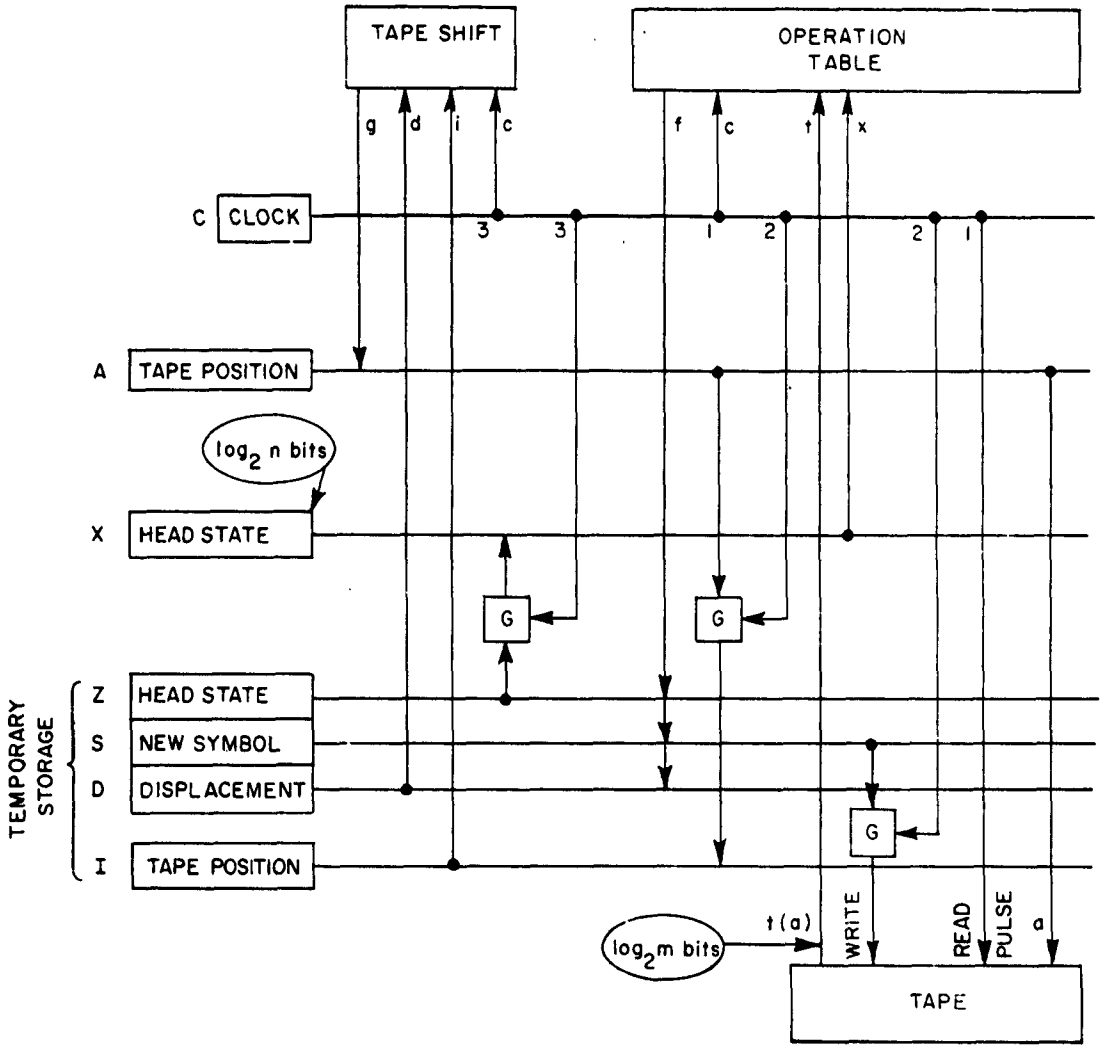
Factual Data

As in the previous report, the work to be reported here will be grouped under three general headings:

1. Synthesis of coding and other digital apparatus
2. Further results on the symbol string algebra
3. Topics concerning digital codes.

1.1 Turing Machines as Stored Program Computers It is desired to study very simple stored program computers because these represent economical ways to synthesize digital transducers. A storage element is usually cheaper than a logic element (diode etc.), especially if the latter must be individually wired. Of course, a stored program computer has the additional advantage that it can be programmed to perform a different task, but if the stored program transducer is cheap enough it can be economically used for a specific application. Instances already exist where people have purchased general purpose computers to accomplish specific tasks rather than design and built their own apparatus. A method for analyzing and synthesizing computers was given in the last report[1]. The basic form was shown in Fig. 9[1], and the timing cycles are shown in Table 1[1]. The purpose here is to throw as much of the burden of computation as is possible on the memory unit. A considerable effort has been expended by various workers in trying to simplify the universal Turing machine, and the resulting designs can be useful as boundary conditions on practical transducers. These Turing machine designs are attempts to reduce the product of member of "tape symbols" and "head states", this product being suggested by Shannon[2] as a suitable measure of complexity. How is the product of tape symbols and head states related to questions of machine operations vs. programmed operations, optimum word length, etc.?

A computer will now be described which resembles a Turing machine except that a finite tape will be assumed. The timing table and block diagram are shown in Fig. 1.1. At time 1 the tape is read and the main calculation is made: the new head state z, the next tape symbol to be printed r, and the tape displacement d are calculated as functions of the old head state x and the tape symbol at a, t(a). At time 2 the new symbol s is printed at position a on the tape and the present tape position is shifted to register I. At time 3 the new head state z is put in the head store X; and the new tape position g is calculated from the old position "a" and the displacement d, thus shifting the tape. The shift in a real Turing machine is either 1 right or 1 left (address increased or decreased by 1); but here, since the tape is finite, more general shifts could be allowed, even to any postiion on the tape (i.e. any address). In a theoretical Turing machine some of the details given in Fig. 1.1 are of no interest, i.e. the clock, the temporary storage, the tape position register and tape shifter. Some of these could be eliminated by postulating suitable delays, eg. if the Operation Table has a delayed output the tape could be read and written one in one part of the cycle. Most Turing machine theory, being concerned with computability and not practical machine design, concentrates on the number of head states n and the number of tape symbols m. As shown in Fig. 1.1, the Head State register will have $\log_2 n$ bits of storage capacity (and that many pairs in the horizontal bus leaving it), and the tape output vertical bus will have $\log_2 m$ pairs. The Operation



	c = 1	c = 2	c = 3
$f[x, t(a)] \rightarrow$	Z	s → T(a)	z → X
	S		
	D	a → I	g(i,d) → A

Fig.I.1

Table thus has $\log_2 mn$ pairs (Boolean variables and their complements) entering it, not counting "c" which is merely a timing pulse from the clock. The Operation Table, if written out explicitly, would have mn entries, each specifying $1 + \log_2 mn$ bits. Using Shannon's method for constructing 2 state or 2 symbol Turing machines[2], and starting from Minsky's machine[4] of 6 symbols and 7 states, the following possibilities exist:

m symbols	n states
2	174
6	7
175	2

These effectively specify the number of Boolean variables entering the Operation Table as between 6 and 9 binary digits. Some thoughts on the design of such multiple output combinational circuits were given in Section 1.2 of the previous report[1]. The parameter $\log_2 m$ in a Turing machine corresponds to word length in a computer. The parameter n in a Turing machine does not correspond to a single parameter in a computer, but it does represent the number of operations and the accumulator size together.

Some interesting comparison can now be drawn between the Turing machine of Fig. 1.1 and the computer of Fig. 9[1]. Words in a single address computer can be broadly divided into three types: operation-data address, operation-instruction address (jumps), and data words (numbers etc.). Instructions are sometimes broken down into more than two fields, eg. operation-index register-address; and the operation sometimes spills over into the address field, eg. in shift instructions. The address may be absolute (directly referring to any part of the memory) or relative (specifying the memory location by its distance from the present instruction). The computer may have many more operations than the operation field indicates, since some can be performed by subroutines. In the extreme case of a Turing machine, the word (tape symbol) cannot be broken down into separate fields at all, and the address is so "realtive" that only the previous or next locations can be addressed. It is desired to study devices halfway between a Turing machine and a single address computer.

The relationship of a Turing machine to various types of digital computer is shown the hierarchy of Table 1.1. The 1, 2, 3, and 4 address machines select their instructions from successive storage locations (except for possible jumps to any part of storage) and their data from arbitrary storage locations given by the address part of the instruction word. The 1 + 1 address machine jumps to an arbitrary location for the next instruction after every step, but this seems to be of no great advantage except where a cyclic storage medium is used[4]. Now it is quite possible to imagine a single address machine to be modified so that the data is part of the instruction word rather than merely the

n Address	DATA			NEXT INST.			Example
	1st Operand	2nd Operand	Result	Usual	Jump	Cond. Jump.	
4	A_1	A_2	A_3	A_4		A_3	SEAC Nat. 390
3	A_1	A_2	A_3	Next	-	A_3	Honeywell 800
2	A_1	A_2	Acc.	Next	A_2	A_1	Univac 1103 A
*	Acc.	A_1	A_2				
	Acc.	A_1	Acc.	A_2		A_1	IBM 650
*	Acc.	-	A_1				
1	Acc.	A_1	Acc.	Next	A_1		IBM 709
*	Acc.	-	A_1				
0.5	Acc.	Pres.Loc. + A		Next	Pres. Loc. + A		CDC 160
	Acc.	-	Pres. Loc. + A				
0	Acc.	Present Loc.	Pres. Loc. Acc.	Next	-	Previous	Turing

* = Store Op's. "Acc." = Accumulator "Ai" = Address field "Loc." = Location

TABLE 1.1

Data Adresses

Instruction		0	1	2	3
Addresses	0	Turing	IBM 709	Univac 1103	Honeywell 800
	1	"Step Data"	IBM 650		National 390 SEAC

TABLE 1.2

address of the data[4]. If the same datum enters several instructions this might prove inconvenient, but it is conceivable that the data would be shorter than the address (obviously true for an infinite tape storage). This suggests the "O-address" computer. A "partial address" is possible as an intermediate step between full address and no a address; eg. in the basic CDC 160[5] has 8192 storage locations and a 6 bit address, relative addresses being used for most operations. Carried to an extreme, the relative address becomes the left or right shift of the Turing machine tape. Specifications for computers mentioned were obtained from references 4 to 8.

An interesting possibility is suggested by Table 1.2 in the box labeled "Step Dat.". This would be the opposite of a 1-address machine. A 1-address machine steps through the instructions while pulling data from various locations specified by the address. A "Step Data" computer would step through the data while pulling instruction from various locations specified by the addresses.

1.2 Basic computer relationships It is desired here to study the relationships between the parameters of a general purpose stored program computer in the same way that Turing machine theory has done for that type. Some of the parameters that should be related are memory size, word size, combinational circuit size, operation time etc. Suppose a computer has a storage unit S consisting of M binary storage elements. Let P, X, and Y be three subsets of S, where P and X are disjoint. Let $\underline{p}_t$ and $\underline{y}_t$ be vectors with binary components representing the contents of the respective memory subsets at time t (t = 0, 1, 2...). The problem which the computer is to solve is to make $\underline{y}_\tau$ a function f of $\underline{x}_o$. Note that $\underline{x}$ and $\underline{y}$ may have many components and $\underline{y}$ may represent the solution to a differential equation, i.e "function" is used generally.

Definition: The program in P calculate f in time τ and places the answer in Y provided that

(1) $\underline{y}_\tau = f(\underline{x}_o)$ for all possible x_o

(2) The result (1) is achieved regardless of the initial states of the elements of S not in P ∪ X.

(3) If the initial value of any single element of P is changed result (1) is not achieved. (This insures that every part of P is necessary at least if the program is to work for all $\underline{x}$.)

(4) For no t less than τ are the above true.

If there are n components in x then there $2^{(2^n)}$ distinct single variable Boolean functions which can be formed from the components of $\underline{x}$. This means that there must be at least

$2^{(2^n)}$ distinct settings of subsets of the M memory elements. Thus

$$2^{(2^n)} \leq 1 + M\,2 + \binom{M}{2} 2^2 + \binom{M}{3} 2^3 + \ldots + \binom{M}{M} 2^M = (1+2)^M$$

$$2^n \leq M \log_2 3$$

If $\underline{y}$ has r components, there are 2^{r2^n} distinct functions, giving

$$r2^n \leq M \log_2 3 \qquad (1.1)$$

The largest memory in use seems to be that of the IBM Stretch computer[8], about 16 800 000 bits. If r = n, this limits n to about 20 bits, certainly much smaller than the input data to most problems. The conclusion is that no computer is "general purpose" is the strict sense.

Nothing has been said so far about the form of the program $\underline{p}$. As is evident from Section 1.1 above, there are a great number of ways in which instruction words can be formed, and a general theory should not put too many restrictions on the way the storage is broken down into words. Note that if part of the program contains the truth table of f, that part alone requires $r2^n$ storage elements. Thus, Eq. 1.1 is a little optimistic in assuming all subset settings are useful programs, but the difference in calculating n is very small. As was suggested in the previous report[1], the program is essentially as complicated as its simplest description (the truth table). This follows when it is realized that the table lookup program could hardly be comparable in size to the complete table of $r2^n$ entries. It appears that no more useful bound than Eq. 1.1 will be found unless the type of problem (Boolean function) is suitably restricted. This same question was raised by Shannon in connection with relay contact networks[9]. His conclusion seems to be that since it requires an enormous number of contacts to realize a randomly selected function of a moderate number of input variables, and since networks have been made with a large number of input variables, that the functions of common interest must be of some special type. The ideas of Shannon on functional separability, group invariance, and symetrical functions seem as appropriate to the computers as to switching networks, the action and complexity of switching networks and computer programs being so similar.

It is not so surprising that even the largest computers cannot calculate an arbitrary function of 20 binary variables when the size simplest description of such a function is

examined. There are $2^{(2^{20})} \approx 2^{(10^6)}$ such functions, so the description requires 10^6 bits. This is about 200,000 English letters, about 3000 lines of type, or about 66 pages of print! This would be 66 pages of compact mathematical description, a description in English would be much longer.

A rough indication of how time of computation might enter the theory can be provided if two additional assumptions are made:

(5) The value of $\underline{x}$ is independent of time during one problem.

(6) The time of execution τ is the same for all problems f.

Now not only must all initial programs $\underline{p}_o$ be distinct, but so must all intermediate states of the initial program region. This leads to

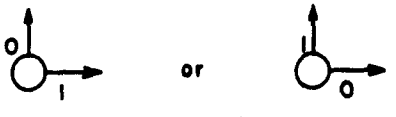
$$r2^n + \log_2 \tau \leq M \log_2 3 \qquad (1.2)$$

For a given r and n the computation time τ cannot be made too large without requiring larger M. This seems contrary to the notion that a sequential computer gains economy by using its parts over and over. However, the bound is obtained for arbitrary functions, for separable or symmetric functions[9] the result might be quite different and long execution times might be indicated. Note that a single table-lookup does not result in a very large value for $\log_2 \tau$.

It is intriguing to consider a computer whose only basic operations are "Sheffer stroke" and "Jump". If $\underline{y}$ is to be a general function of $\underline{x}$, it is only necessary to consider r separate scalar functions y_i of $\underline{x}$. Each of these can be expressed in normal form as a logical sum of logical products (of the variables and their negatives). Each product of n terms can be obtained by successively multiplying just 2 terms, and similary for sums. But each of the 3 Boolean operations (sum, product, negate) can be expressed in terms of the single binary operation Sheffer stroke $X|Y$ (either not X or not Y)[10,11]. The stroke operation would suffice not only in handling the data $\underline{x}$ but also in bookeeping operations as the program itself is modified then $\underline{p}_o, \underline{p}_1 \cdots \underline{p}_\tau$. For ordinary computer programs the expression of all operations (say addition of two 10 digit numbers) in terms of the stroke function would result in a great increase in complexity. However, if subroutines and symbolic programming are used, the program writing task might not be too bad. The control and "arithmetic unit" parts of the computer would be very simple, as in a Turing machine. The problem remains, however, that if nothing is done to limit the type of function which the computer is to handle any interchange between control and "arithmetic unit" complexity and storage capacity will be masked by the size of the truth table.

1.3 Tabulation of form state binary machines In a previous report[12], a tabulation of these state binary machines was made, and this has proved very useful in giving examples of machines with certain combinations of properties (eg. non-resettable, non-periodic and simple). An n state binary machine will be defined here as a collection of n states with a O arrow and a 1 arrow leaving each state and ending in a state, possibly the same state. The machine is to be strongly connected, ie. every state must be reached from every other state.

Consider the graph shown in Fig. 1.2a. This is topologically the same as that shown in Fig. 1.2b. Also, for the purposes of this enumeration, Fig. 1.2c is considered the equivalent of the other two. It is desired to tabulate only topologically distinct graphs, so that different orientations, reflections, notations etc. do not yield new cases; and if all the labels on the arrows are simultaneously changed, the resultant graph is not considered distinct. The big problems in making such a tabulation are to include all possible graphs and not to include two equivalent graphs. The method adopted is to start with a simplified graph, and then to add details in such a way that each graph can be formed in one and only one way. As shown in Fig. 1.3, the graph of Fig. 1.2a is the fifth version, the steps in adding loops, adding multiple connections, adding directions to the connections and finally, labeling the arrows. The number of four state diagrams found at each step (counting only those leading to possible find graphs meeting the required conditions), are 5, 24, 108, as shown in Fig. 1.3, and there are 460 graphs in the final tabulation. There are actually six graphs of the type shown in Fig. 1.3a, but one (F) does not lead to any (binary) graphs meeting the conditions. The number of distinct graphs of each type are shown in Fig. 1.4. As an example of how the classification is carried out, Fig. 1.5 shows the variants of D. Consider D_3, there are already 8 transitions so that no double connections can be added. The arrows can only be drawn in two ways shown in Fig. 1.6a and b. Now the first arrow can be labeled either I or 1, so label the diagonal arrow in all cases O. This immediately requires one other arrow to be labeled 1 in each graph. Fig. 1.6a has no symmetrics, so that the remaining 3 arrows (3 being always oppositely labeled) can be labeled in 8 different ways. On the other hand, it does not matter in Fig. 1.6b whether the lower left state is labeled



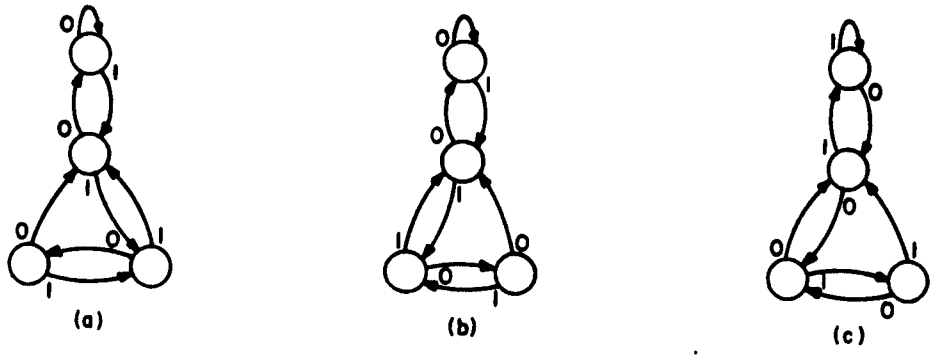


Fig. I.2 Equilalent Graphs

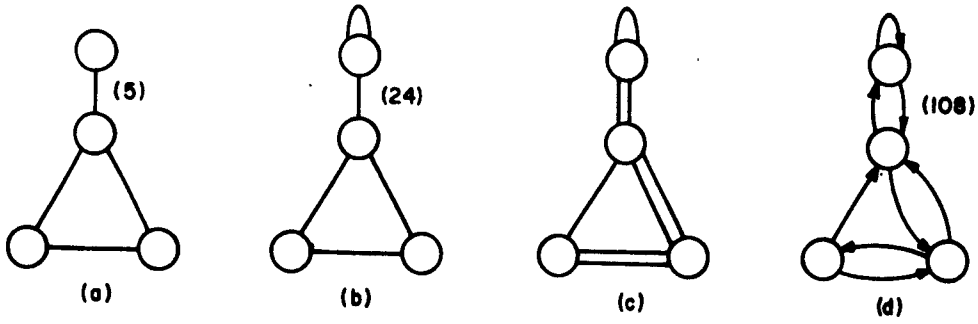


Fig. I.3

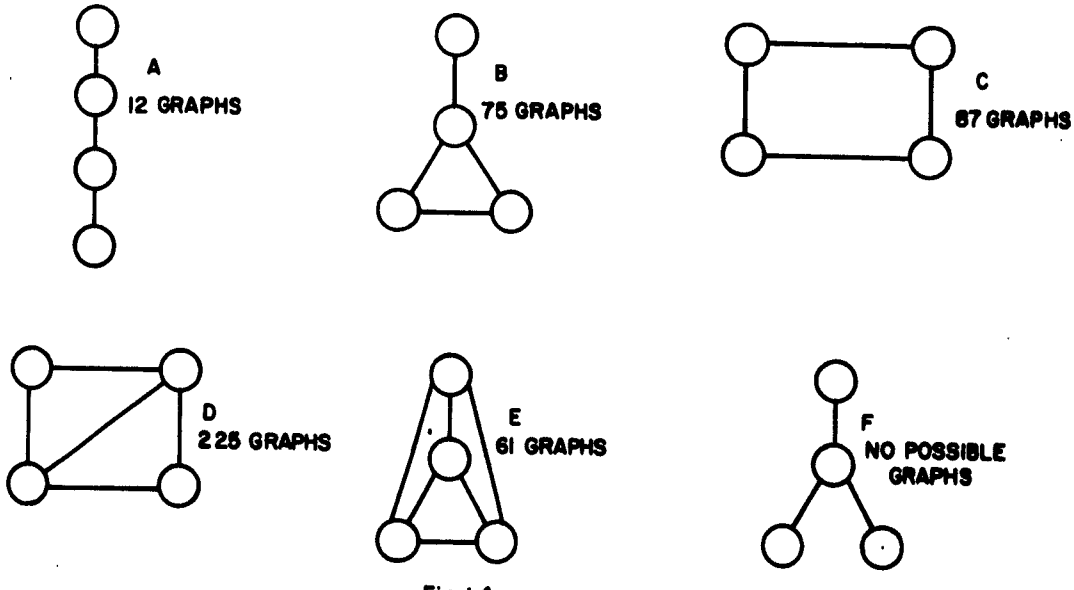


Fig. I.4

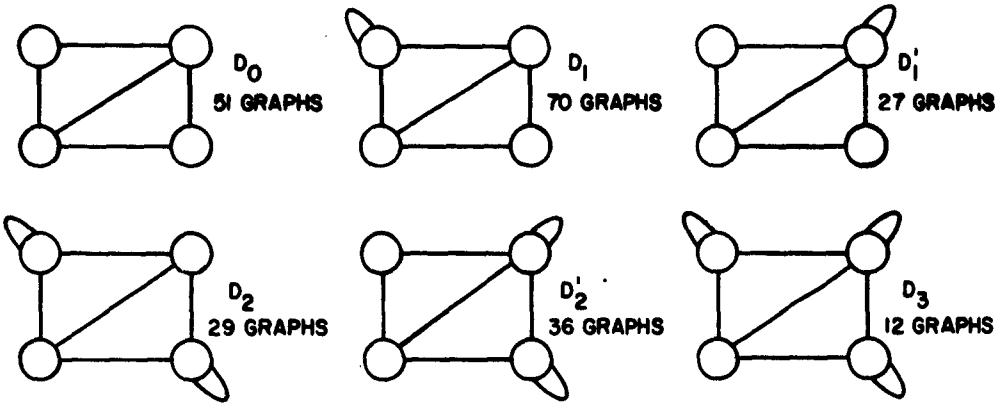

D_0
51 GRAPHS
D_1
70 GRAPHS
D_1'
27 GRAPHS
D_2
29 GRAPHS
D_2'
36 GRAPHS
D_3
12 GRAPHS

Fig. I.5

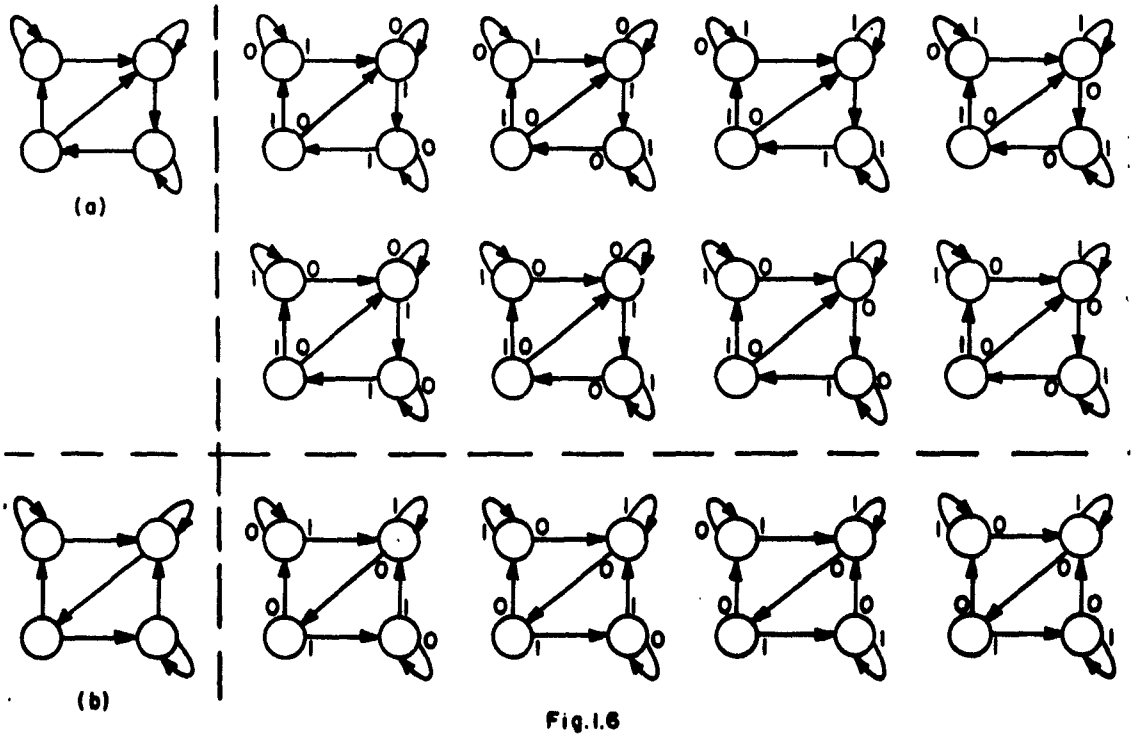

(a)
(b)

Fig. I.6

because of symmetry, therefore only 4 graphs result from Fig. 1. 6b. The 108 labelless directed graphs such as in Fig. 1. 6a, b can all be labeled in 8, 6, 4, 3, 2, 1 ways according to the types and numbers of symmetries they possess.

Four states can be represented by the settings of a pair of flip-flops, so that there are 460 ways to feed a single binary input to a pair of flip-flops; each resulting in a different behavior pattern, but each allowing every state to be reached from every other. If this pair of flip-flops is to be connected into a layer network interchanging the O's and 1's at the input and relabeling the states can result in different behavior of the larger network, and if these changes are considered to give distinct machines the total number possible will be somewhere between 460 and 2 x 4 ! x 460 types. Although symmetries will keep the number considerably below the upper bound, the number of ways of connecting just two flip-flops is remarkably large.

The tabulation of the 460 graphs was done some time ago but was never published because it was not sufficiently checked. Recently this number has been checked by a thesis student, Mr. Donald Chin. A tabulation of the 460 types will be prepared as a separate report.

2.1 Semigroups without the descending chain conditions In dealing with the semigroup corresponding to symbol concatenation in languages, it is usually assumed that the semigroup is free (no relations between the generators), or that it satisfies the descending chain condition (any sequence of strings obtained by cancelling symbols from beginning or end must terminate). These conditions are equivalent and they imply a unique factorization of any string into symbols. In dealing with compound code the semigroup symbols are matrices, the descending chain condition does not hold, and the factorization is not unique. In the Third Semi-annual report[13], the descending chain condition was replaced by Postulate 14[13], that every element have some finite factorization. It is desired here to study the structure of semigroups having neither Postulate 14[13] nor the descending chain condition, but which nevertheless allow certain canonical factorizations.

Consider a set of elements S with a closed binary operation (indicated by juxtaposition) obeying the postulates P 1 to P 6 below.

P 1: For every x, y, and z in S, $x(yz) = (xy)z$. A zero will be difined as an element θ which obeys $\theta x = x\theta = \theta$ for all x in S.

P 2: For every x, y, and z in S, $xy = xz$ implies $y = z$, provided that $xy \neq \theta$. Similarly $yx = zx$ implies $y = z$, provided that $yx \neq \theta$.

P 3: For every w, x, y, and z in S, $wx = yz \neq \theta$ implies that either a u exists in S,

such that $wu = y$, or that a v exists in S such that $w = yv$. The dual statement can be taken as a postulate or proved from the other postulates.

P4: Each element has a right and left unit, possibly depending on the element. That is, for each x in S there exists some u in S such that $xu = x$ and some v in S such that $vx = x$. An element e, not equal to θ, will be difined as idempotent if $ee = e$. The following lemmas will be useful in proving the main theorems:

L 1: The zero, if it exists, is unique. Suppose θ_1 is a unit. Then $\theta_1 x = \theta$ for all x, including any other unit θ_2. But if θ_2 is a unit $\theta_1\theta_2 = \theta_2$. Therefore $\theta_1 = \theta_2$.

L 2: Similarly, there is no element $w(\neq \theta)$ such that $wx = \theta$ for all x in S. By P4, there exists a u such that $wu = w$, but multiplication is unique. Note that there may be divisors of zero, ie. pairs of elements x, y such that $xy = \theta$.

L 3: For any particular element x in S, the right unit is unique and the left unit is unique. Proved by P 2.

L 4: Every idempotent is the unit of some non-zero element; and every unit of a non-zero element is idempotent. For every idempotent e is the unit of itself ($ee = e$); and if u is a unit of $x(\neq \theta)$ it follows that $x = xu = xuu$, therefore $u = uu$ by P 2.

L 5: The product of two distinct idempotents is zero. For $e_1e_2 = e_1e_2e_2$ implies either $e_1e_2 = \theta$, or by P 2, $e_1 = e_1e_2$. But $e_1 = e_1e_1$, therefore $e_1 = e_2$ by L 3.

L 6: The set of elements eS, where e is an idempotent of semigroup S, is a non-empty subsemigroup of S. It is non-empty since S contains the right unit of e and eS then containts at least e. Let es_1 and es_2 be two elements of eS, then $(es_1)(es_2) = e(s_1es_2) \in eS$.

L 7: Let e_1 and e_2 be two distinct idempotents of S, then e_1S and e_2S have no common elements except θ. Suppose $e_1x = e_2y \neq \theta$ where x and y are elements of S. By P3 either there exists a u in S such that $e_1u = e_2$, or a v in S such that $e_1u = e_2$, or a v in S such that $e_1 = e_2v$. In the former case, multiply both sides by e_1: $e_1u = e_1e_2 = \theta$ (by L5), then $e_2y = \theta$ contradicting the original hypothesis. A similar argument applies to the second possibility.

L 8: Every element of S is in one of the subsemigroups e_iS. Consider any element x of S. Its left unit is an idempotent by L 4. If e_j is the left unit of x, then $e_jx = x$ and x must be in e_jS. Lemmas 6 to 8 apply dually to the subsemigroups Se.

L 9: The class e_iSe_j is a subsemigroup and it is disjoint with e_kSe_h unless $i = k$ and $j = h$. Every element of S is in one of e_iSe_j. Also e_iSe_j $(i \neq j)$ is a null semigroup,

ie. all products are 0. Proof similar to above.

The lemmas presented above show that S can be expanded into disjoint semigroups in several ways:

$$S = \sum_i e_i S \qquad \text{(right ideals)} \qquad (2.1)$$

$$S = \sum_j S e_j \qquad \text{(left ideals)} \qquad (2.2)$$

$$S = \sum_{i,j} e_i S e_j \qquad (2.3)$$

These are analogous to expanding a group into cosets. An expansion into two sided ideals can be obtained by applying Eq. 2.1 to S = SS:

$$S = \sum_i S e_i S \qquad (2.4)$$

Now, in order to prevent the semigroup from consisting of many subsemigroups entirely disconnected with each other, assume:

P 5: For every e_i and e_j there exists elements ϕ_{ij}, ϕ_{ji} of S such that

$$e_i = \phi_{ij}\, e_j\, \phi_{ji}$$

Theorem 1 The following expansion of the semigroup S holds:

$$S = \sum_{ij} \phi_{ik}\, S_{kk}\, \phi_{kj} \qquad \text{(for any k)} \qquad (2.5)$$

where $S_{kk} = e_k S e_k$

$$\phi_{ii} = e_i$$

This theorem is easily proved using P 5 and Eq. 2.3. It follows that every element s in S has an expansion

$$s = \phi_{ik}\, s_{kk}\, \phi_{kj} \qquad (2.6)$$

where there are several choices for s_{kk} but where i and j are uniquely determined. The subsemigroups $\phi_{ik}\, S_{kk}\, \phi_{kj}$ are isomorphic for different k, i and j fixed; and are disjoint, Finally:

P 6 The subsemigroup $S_{11} - e_1 - 0$ is freely generated by $g_1, g_2 \ldots g_n$. It follows

now that any element of S can be expanded uniquely according to Eq. 2. 6 as

$$s = \phi_{11} \xi_1 \xi_2 \cdots \xi_\ell \phi_{1j} \qquad (2.7)$$

where $\xi_h = g_1, g_2 \cdots g_n$ and ℓ is the "length" (possibly 0) of s.

Theorem 2: In S there are only 5 types of elements:

1) the zero θ
2) the idempotents e_i
3) the ϕ_{ij} $(i \neq j)$
4) the generators $g_1, g_2 \cdots g_n$
5) composite elements as in Eq. 2. 7.

The abstract formulation given above can be clarified by noting that θ is a matrix of empty sets; the elements ϕ_{ij} consist of a matrix with a null string in the i'th row and j'th column and the other elements empty sets; $e_i = \phi_{ii}$ that is a null string on the diagonal; the generator g_i consist of a single letter in the first row, first column and empty sets elsewhere; and composite elements have a single string in one position of the matrix and empty sets elsewhere. The whole semigroup S is the set of union-irredicible elements[12]. The abstract formulation is given in an attempt to obtain a mathematical system which is neither too specific nor too general, and therefore to obtain a one-to-one correspondence between coding theorems and algebraic theorems. By this means it is hoped to extend theorems about simple codes to compound codes, as was done for the Sardinas-Patterson algorithm[14].

2. 2 Further results in the code string algebra The code string algebra which was developed in previous reports[13] deals with matrices of sets of symbol strings, allowing the null string ϕ. Let * be any binary closed operation which distributes over addition + (set union, position by position in the matrix). If $G \subseteq H$ is defined to mean that there exists a (possible empty) matrix X such that $G + X = H$, then

$$G \subseteq H \text{ implies } P * G * Q \subseteq P * H * Q \qquad (2.8)$$

because $P * G * Q + Y = P * H * Q$ where $Y = P * X * Q$. Here * might stand for · (contatenation or multiplication), or \, or / (divisions). Similar reasoning show that

$$G \subseteq H \text{ and } I \subseteq J \text{ implies } G * I \subseteq H * J \qquad (2.9)$$

The relation between * and intersection can now be established: First note that since $G \cap H \subseteq G$, by Eq. 2. 9 it follows that $P * (G \cap H) * Q \subseteq P * G * Q$. Next note that $A \subseteq B$ and $A \subseteq C$ implies $A \subseteq B \cap C$. Set $A = P * (G \cap H) * Q$, $B = P * G * Q$, and $C = P * H * Q$

and it follows that

$$P * (G \cap H) * Q \subseteq (P * G * Q) \cap (P * H * Q) \qquad (2.10)$$

again where * represents either concatentation or division. Of course, one sided relations can be obtained from all of the above equations by setting either P or Q equal to a diagonal matrix of null strings.

In ordinary matrix theory (elements of the matrix in a commutative ring), the law $(AB)' = B'A'$ holds, where prime indicated the transpose. This is not true in the code string algebra since the semigroup of element multiplication is not commutative. If T (ϵJ) a matrix with one null string in every row and one null string in every column, and if Ψ is a matrix with a null string in every position on the main digonal, then

$$T\,T' = T'\,T = \Psi \qquad (2.11)$$

$$T_1\,T_2 \in J \quad \text{if} \quad T_1 \in J \text{ and } T_2 \in J \qquad (2.12)$$

These follow from the fact that the T matrices represent permutations. Now it follows that

$$\left.\begin{array}{l} A = T\,X\,T' \\ B = T\,Y\,T' \end{array}\right\} \quad \text{implies} \quad AB = T\,X\,Y\,T' \qquad (2.13)$$

$$A = T\,X\,T' \quad \text{implies} \quad A^n = T\,X^n\,T' \qquad (2.14)$$

A matrix A will be defined as automorphic if $A = T\,A\,T'$ with $T \neq \Psi$. Any power of an automorphic matric is automorphic by eq. 2.14.

2.3 Resettability of finite state machines In another report [12] there have been tabulated the state diagrams with 1, 2 and 3 states such that exactly two arrows (labeled 0 and 1, the machine inputs) leave each state and such that every state can be reached from every other state. Only topologically distinct graphs are shown, ie. graphs obtained by interchanging the labels on all arrows simultaneously, or by premuting the states, or both are not shown. Under the restrictions it was found that there is 1 graph of 1 state, that there are 4 graphs of 2 states, and 29 graphs of 3 states*. The graphs have been classified according to 3 properties (simple or compound, resettable or not, bounded delay or not) thus producing 8 categories. It is necessary to go to 4 state graphs in order to give examples in 2 of these categories, this is done on the next page of the report[12]. A graph is resettable if and only if a reset signal exists; a reset signal is a sequence of 0's and 1's which will put the machine in a single definite state no matter

*Some errors have been found in this tabulation: graphs 14, 15 and 32 have 2 components and not 3; graph 5 has the lower left arrow in the wrong direction.

what the starting state is. The concept of resettability is quite similar to the concept of ergodicity in physics; if a resettable machine is fed with a random input sequence of O's and 1's the effect of the starting state on the probability of the present state gradually "wears off" as the length of the input sequence increases. It is desired to obtain necessary and sufficient conditions for a machine to be resettable. As a first step, the reasons why 12 of the 36 machines in the old report[12] are not resettable will be examined. The string set matrix[14] of the machines under consideration will have a single O and a single 1 in every row, and will have no null strings. Here the strings represent inputs, there is a single O in each row because the machine must know how to change state and can only go to one state, and there are no null strings since they would represent spontaneous state changes. Consider various powers of the matrices of 2 states:

$$M_{18} = \begin{bmatrix} 1 & 0 \\ 0 & \\ 1 & \end{bmatrix} \qquad M_{18}^2 = \begin{bmatrix} 00 & \\ 01 & 10 \\ 00 & \\ 01 & 00 \\ 11 & 10 \end{bmatrix}$$

$$M_{21}^2 = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \qquad M^2 = \begin{bmatrix} 00 & 01 \\ 10 & 11 \\ 00 & 01 \\ 10 & 11 \end{bmatrix}$$

$$M_{24} = \begin{bmatrix} & 0 \\ & 1 \\ 0 & \\ 1 & \end{bmatrix} \qquad M_{24}^2 = \begin{bmatrix} 00 & \\ 01 & \\ 10 & \\ 11 & \\ & 00 \\ & 01 \\ & 10 \\ & 11 \end{bmatrix} \qquad M_{24}^3 = \begin{bmatrix} & 000 \\ & 001 \\ & 010 \\ & \cdots \\ & 111 \\ 000 & \\ 001 & \\ 010 & \\ \cdots & \\ 111 & \end{bmatrix}$$

$$M_{28} = \begin{bmatrix} 1 & 0 \\ & \\ 0 & 1 \end{bmatrix} \qquad M_{28}^2 = \begin{bmatrix} 00 & 01 \\ 11 & 10 \\ 01 & 00 \\ 10 & 11 \end{bmatrix} \qquad M_{28}^3 = \begin{bmatrix} 001 & 000 \\ 010 & 011 \\ 100 & 101 \\ 111 & 110 \\ 000 & 001 \\ 011 & 010 \\ 101 & 100 \\ 110 & 111 \end{bmatrix}$$

The codes M_{18} and M_{21} are resettable, but M_{24} and M_{28} are not. The n'th power of the code matrix shows the final state (column) as a function of the initial state (row) and the input signal of n symbols. The code is resettable if and only if one signal appears in every row of some column (possibly accompanied by other signlas) in some power of the matrix. If such a signal does appear every row of some column of the n'th power, there will then be a signal appearing in every row of some column of the n'th power where $n \geq n_o$. In fact, once one column has such a reset signal in it, all columns can be made to contain resets by increasing n. The smallest n_o for which resets appears gives the length of the shortest reset signal, $n_o = 1$ in the above examples. A few more examples will be given:

$M_{25} =$

	0	1
0		
1		
0		
1		

$M_{25}^2 =$

00			
01			
10			
11			
	00	01	
	10	11	
	00	01	
	10	11	

$M_{25}^3 =$

	000	000
	001	001
	...	...
	111	111
000		
001		
...		
111		
000		
001		
...		
111		

This is a periodic machine, and hence not resettable. Note that the disjoint patterns of empty positions periodically repeat themselves so that no column can ever have a member in every row.

$M_{13} =$

	0	1
0		
1		
0		1

$M_{13}^2 =$

00		
01		
10		11
	00	01
	10	11
		01
10	00	11

Here $n_o = 2$ and 11 is the reset, appearing first in the third column.

It is desired to find general algebraic conditions for resettability. In the following, $\mathfrak{C}$ will be used to denote the class of single column matrices, and C will always mean some member of that class, eg. $\begin{bmatrix} 110 \\ 110 \\ 110 \end{bmatrix}$. The machine M is evidently resettable if and only if there exists an $n = 1, 2 \ldots < \infty$ and a $C \in \mathfrak{C}$ such that

$$C \subseteq M^n \tag{2.15}$$

where inclusion and matrix multiplication are defined as in previous reports[14]. Some conditions on M which present M from being resettable can now be established. According to Section 2.2 above if M is automorphic, ie.

$$M = T\,M\,T'$$

then M^n will be automorphic and

$$C \subseteq T\,M^n\,T'$$

Now by Eq. 2.8 above,

$$T\,C\,T' \subseteq T\,M^n\,T'$$

But T C T' has the same string as C but in a different column, and therefore both cannot be in the same matrix if the matrix represents a determinate machine. An automorphic matrix corresponds to a machine with a proper automorphism, that is the machine looks the same a permutation of the states. If the state diagram looks the same after permuting the states, any reset signal would have the same effect on the graph as on the permuted graph. The existence of automorphisms is then the same as the existence of certain types of symmetry in the state diagram.

Certain other cases of non-resettability can be recognized. A state diagram is defined as periodic if the greatest common division of the path lengths which start at a fixed state and return to that state is greater than unity. If this g. c. d. is equal to p, then the graph has a period p and the states can be divided into p classes $K_1, K_2 \ldots K_p$ such that either a 0 or a 1 causes a transition from K_i to K_{i+1} (or from K_p to K_1). This is shown in the analysis of Markov precesses, eg. Feller[15], page 330. A periodic machine arises if a uniform code is decoded, and this is one of the classes of non-ergodic codes given by Schützenberger[16].

The next class of non-resettable graphs arises from the fact that it is necessary to have at least one state with two or more 0 arrows entering, or at least one state with two or more 1 arrows entering. If no such state exists, then the last digit of the reset signal cannot bring together the final ambiguous class of states. But if no state with at least two similarly labeled arrows entering it exists, then every state must have just two arrows entering it, a 0 arrow and a 1 arrow. This is true because the total number of arrows is equal to twice the number of states, and a single arrow entering one state implies three or more entering another. The particular type of state diagram considered here might then represent a binary determinate machine if all arrows are reversed in direction. Since it makes sense going backward, such a graph will be called a palindrome. Note that it may or may not represent exactly the same machine if the arrows are

reversed; numbers 29 and 32 in Fig. 30[12] do represent the same machine. The idea of reading backward occurs in a palindrome and in the anagrammtic codes of Schützerberger[16] but the decoder of a simple code with the prefix property cannot be a palindrome since all words end in the base state. It is easy to see why a palindrome cannot satisfy Eq. 2.15 above. In a palindrome a 0 input permutes the state, and so does a 1 input. Th refore, the M matrix is merely the sum of two T-type matrices (single element in each row and column), one for 0 and one for 1. Any power of M will be a sum of T-type matrices:

$$M = T_0 + T_1$$

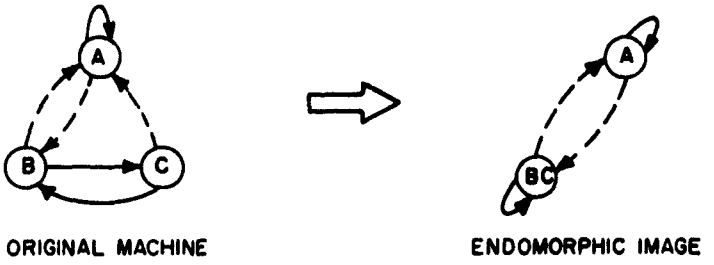
$$M^2 = T_0T_0 + T_0T_1 + T_1T_0 + T_1T_1$$

...

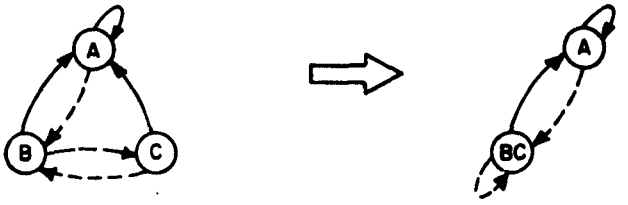
Since each input string appears in only one terms of this sum, there cannot be a column of identical strings.

The third class of non-resettable machines will be those in which an endomorphic image is non-resettable. Here an endomorphism is a mapping from the set of states to a proper subset of states such that the endomorphic image has a single 0 arrow and a single 1 arrow and a single 1 arrow leaving each state. In Fig. 30[12], number 27 has the following endomorphism: A→A, B→B, C→B. This is shown in Fig. 2.1a Some periodic machines are a special case of this class, since they have an endomorphic image which is a closed circle of p states (a palindrome). A resettable machine with a resettable endomorphic image is shown in Fig. 2.1 b. Examples of machines with no endomorphic images except the single state machine are Fig. 30[12] number 6 (resettable) and Fig. 30[12] number 23 (non-resettable, a periodic circle). There are non-resettable machines in which all endomorphic are resettable images, eg. Fig. 31[12]a.

The three causes of non-resettability given above account for all cases involving two and three state machines, but examples of four and five state machines can be given which are not periodic, not palindromes, and which have no non-resettable endomorphic images. The first example is related to the uniformly composed codes given by Schützenberger[16] as one of the classes of non-ergodic codes. A uniformly composed code can be derived by replacing the symbols in a non-binary code by the binary words of unequal length.

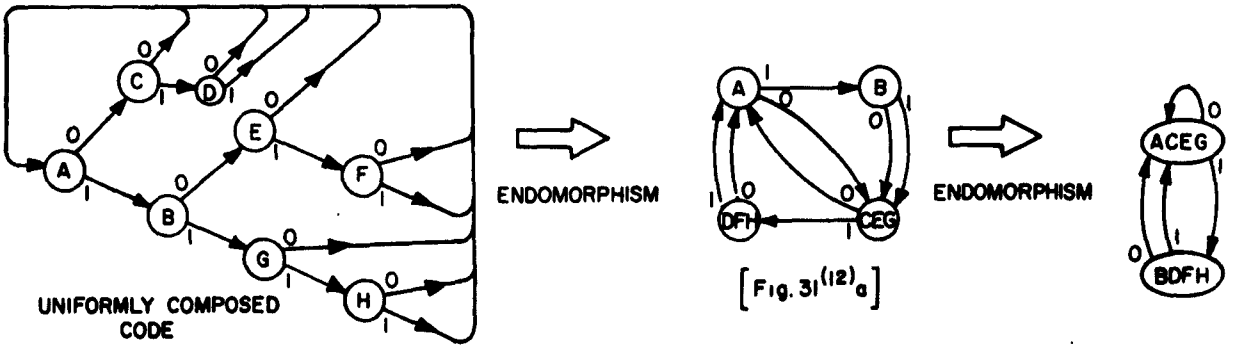


(a) NON-RESETTABLE MACHINE (M27)



(b) RESETTABLE MACHINE (M19)

Fig. 2.1 ENDOMORPHISMS



(A=X1, B=Y1, C=X4, D=Y4 ETC.)

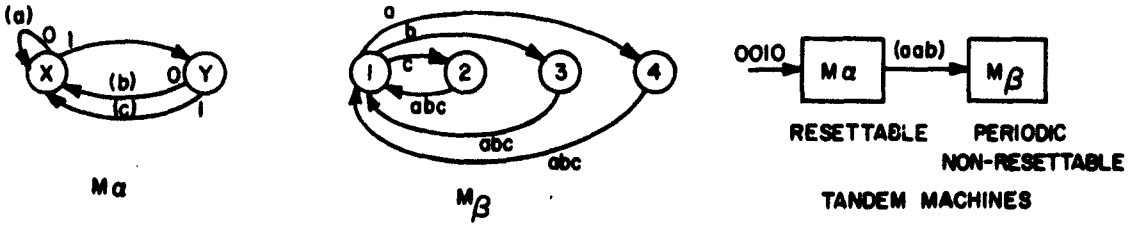


Fig. 2.2 Uniformly Composed Code

a a
a b
a c
b a
b b
b c
c a
c b
c c

with the replacement a= 0, b= 10, c = 11. Even if the set of binary words is such that the ternary letters can be identified, there will never be any way of knowing whether the ternary letter is the first or last in one of the nine words above. The example given in the coding report in Fig. 31[12] a can be derived from Schützenberger's example as shown in Fig. 2.2. The four state example can be derived more directly and shown in Fig. 2.3. It might be thought that non-resettability might arise in more complex ways by hiding a non-resettable machine in a complex network of resettable machines, as in Fig. 2.4. This can be put in the form of Fig. 2.3 by grouping the components if there are no feedback loops, as shown by the dashed lines. If the input feeds the non-resettbble machine directly, as in Fig. 2.5, there will be a non-resettable endomorphic image. The type of non-resettability described above will be referred to as due to a composite machine having a non-resettable component.

A non-resettable machine which fits into none of the above four categories is shown in Fig. 2.6. The fact that this machine has a prime number of states does not rule out its being a composite machine since a transient state is possible, but if it were composite there would be an endomorphic image. The machine of Fig. 2.6 is an endomorphic image of the machine of Fig. 2.7, which is the decoder of the anagrammic code of Schützenberger[16]:

0 0 0
0 0 1 0
0 0 1 1
0 1
1 0 0
1 0 1 0
1 0 1 1
1 1 0
1 1 1

This code is complete and has the prefix property both from the left and from the right. Now Fig. 2.6 can be obtained from Fig. 2.8 by the coalescing process shown in Fig. 2.9. Here it is necessary to find a pair of states such as X_1 and X_2 whose 0 arrows go to the same Y and whose 1 arrows go to the same state Z. Note that X, Y, a, b, c, d, e, need

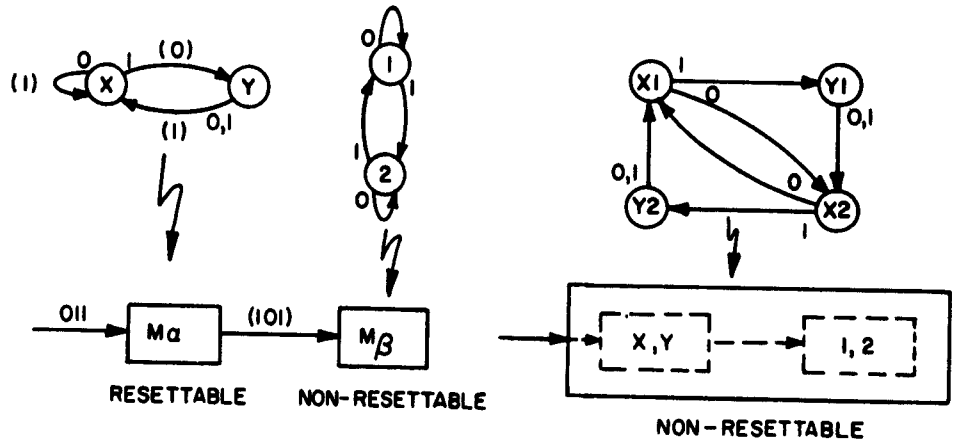


Fig.2.3 Tandem Machines

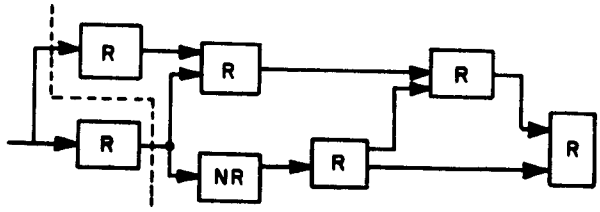


Fig.2.4 Machine with a Non-Resettable Part

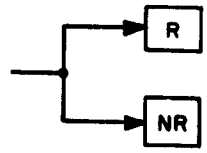


Fig.2.5

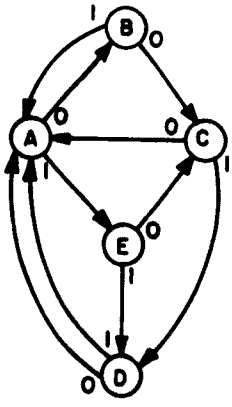


Fig.2.6

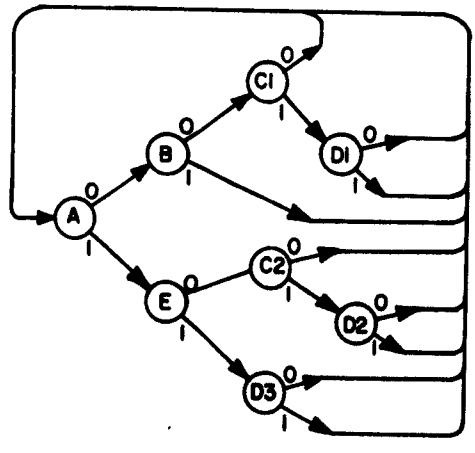


Fig.2.7 Anagrammatic Decoder

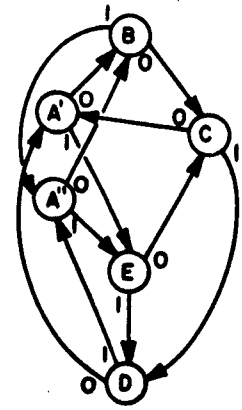


Fig.2.8

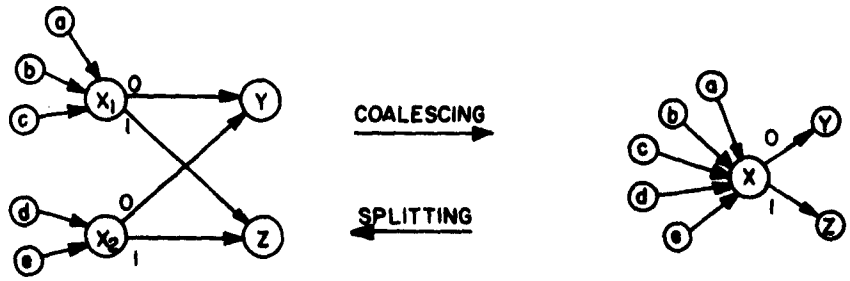


Fig. 2.9 Coalescing Process

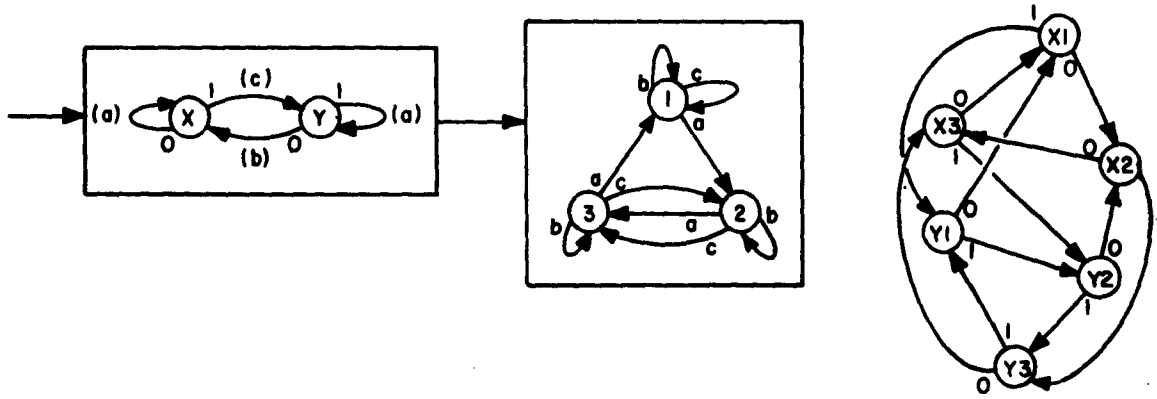


Fig. 2.10 Tandem Machines From Anagrammatic Code

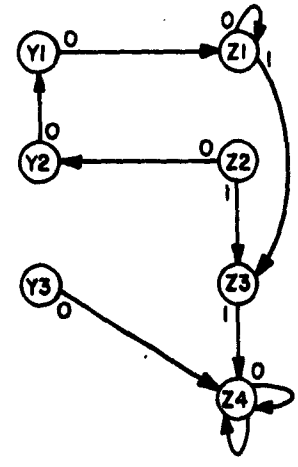


Fig. 2.11 Resettable Graph

not be distinct from each other or from X_1 and X_2. The converse process of splitting a state into two states can always be carried out. Now it is obvious that the property of resettability is invariant to coalescing or splitting. To show this note that if a reset sequence for state Y exists in Fig. 2.9 on the right machine it also will reset the left machine to Y, and conversely. If no such reset signal exists for one machine, none exists for the other. A machine M' which can be obtained from a machine M by repeated coalescings will be said to be a contracted form of M, and M will be said to be an expanded form of M'. A contracted machine is an endomorphic image, but not every endomorphic image can be obtained by coalescing. Since Fig. 2.6 is a contracted form of the non-resettable Fig. 2.8, it cannot be resettable. It remains to show that Fig. 2.8 is non-resettable. In Fig. 2.10 a tandem connection of a binary resettable machine and a ternary non-resettable machine (palindrome) is shown to be equivalent to Fig. 2.8.

There are several sufficient conditions for resettability which are known. For example, the set of states marked Z_1, Z_2, Z_3, Z_4 in Fig. 2.11 has the property that a sequence of 1's eventually makes the state Z_4. If all states are in such a set the graph is naturally resettable. If a sequence of 0's results in the only possible states being in one of the Z's, then the graph is also resettable, and so forth.

The problem of resettability does not seem to have been treated in the literature of finite state machines. Ginsburg describes some similar problems in his book[18], but very strong assumptions are made concerning the existence of output signals from the machine.

3.1 Testing for resettability The only algorithm which has been mentioned here for determining whether or not a particular finite state machine is resettable or not is that implicit in Eq. 2.15, and this becomes an algorithm only when a bound is found for the length of a reset signal. A non-algebraic algorithm will now be given which will provide such a bound, and which is very useful for visualizing the resetting process. Consider n identical machines fed by the same input (where n is the number of states) but each started in a different state. The composite machine will have a state diagram which will show at a glance whether the original machine is resettable and what the shortest reset signal is. The algorithm will be illustrated using the machine M_{13} of Fig. 30[12] of the old report and Section 2.3 above. Refer to Fig. 3.1 and note that the state "ABC" might be thought of as either "A and B and C" (parallel machines) or as "A or B or C" (indicating complete uncertainty as to the state). After the input "0" the state C becomes impossible and therfore "AB" results. The resulting composite graph has $n + \binom{n}{2} + \binom{n}{3} + \ldots + 1 = 2^n - 1$ states, and the graph can be finished until a 0 and a 1 arrow leave each state. The original graph will appear as part of the

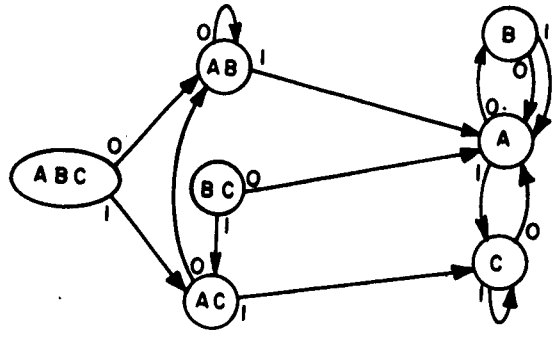


Fig. 3.1 Resettable

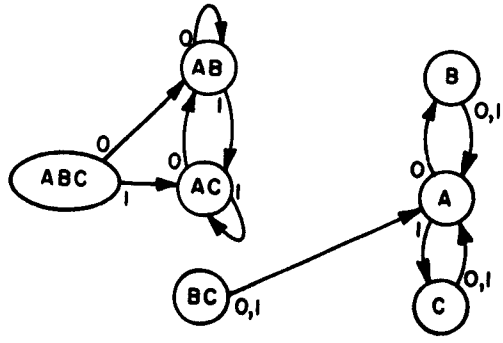


Fig. 3.2 Nonresettable

composite graph, those "single letter" states. If a path is found from the composite state representing all states to any state of the original graph, then the graph is resettable. The shortest path gives the shortest reset signals. Since the machine is determinate, there will be no path from a state with r letters to a state with more than r letters, and in particular no arrows leave the original graph. In Fig. 3.1 it can be seen that the shortest reset signals are 01 and 11, and that the shortest signal which insures state B is 010. Similarly, a signal which helps the machine continually in an unknown state is 10000.... In Fig. 3.2 in shown a graph which is not resettable, the periodic machine M_{25}. Palindromes have both arrows returning to the all-state node:

In a bounded delay machine[12], all sequences longer than a certain amount are resets. It is now apparent that the longest path from the all-state to a single-state can have no more than $2^n - n - 1$ symbols.

In Fig. 3.2 it will be noticed that starting from ABC, a closed set{AB, AC}is reached. Every determinate connected machine will have one and only one such a closed set which can be reached from the all-state. This closed set will have the following properties:

1) all state labels have the same number of letters

2) every letter (original state) appears at least once in the closed set

Let the closed set have labels with λ letters. If $\lambda = 1$ the machine is resettable, if $\lambda > 1$ it is not resettable. If $\lambda = n$ the machine is a palindrome, as defined in Section 2.3 above.

3.2 Calculation of code "compression" The coding process which is to be considered here is sometimes called "recoding": a sequence of discrete symbols from an information source is decomposed into message words, these words are transformed into signal words by the encoder and transmitted over the channel, the decoder decomposes the sequence of signal words and transform then back to message words, and finally, the message words are strung together to form the original source sequence. The "compression" achieved by the code is the ratio of the length of a source sequence to the length of the corresponding channel sequence, provided that the same size alphabet is used in each sequence. Since the ratio referred to depends on the length of the sequence and on the particular words in the sequence, the definition must be made more explicit. It is the purpose of this section to compare several formulas for calculating this

compression. If the set of message words is fixed and their occurrence statistically independent, Huffman[17] has shown how to maximize the compression ly properly chosing the channel words. If the whole recoding process is considered there will not generally be a single most efficient code because the longer the words (and the more of them there are) the more compression will be achieved. There may be, however, a maximum compression for codes of a given "complexity". The complexity of a code should be defined in terms of the size of the apparatus required to encode and decode it, and an exact method for calculating complexity necessarily involves a method of sequential apparatus synthesis.

The amount of compression achieved by a code depends not only on the code, but on the information source. The sources to be considered here will be finite Markov processes with fixed transition probabilities. The encoder does not know the internal states of the source during the process, but there is a binary word associated with each state transition and this is fed to the encoder. The source can be assumed to be in the following standard form without loss of generality: There are m internal states S_1, S_2, ... S_m. From each state there are exactly two transitions possible, one generating a 0 and the other a 1 to be fed the encoder. These two transitions can be represented by arrows in the state diagram used to visualize the process; an arrow leaving one state can either go to another state or return to the same state. Let p_{ij} be the probability of the i'th state being next if the present state is S_j, and let r_{kj} ($k = 0, 1$) be the probability of the symbol k being generated by the source if it is presently in state S_j. It follows that p_{ij} is either zero, equal to one of the r_{kj}, or equal to unity. It will be assumed that the source is ergodic[15] so that a unique set of state probabilities P_i exists satisfying

$$\sum_{j=1}^{m} p_{ij} P_j = P_i \qquad i = 1, 2 \ldots m \qquad (3.1)$$

The encoding process will be described by a finite state transducer having states R_1, R_2, ... R_n. Again two arrows will leave each state, one for a 0 symbol from the source and the other for 1. Associated with each arrow is either a channel word of finite length, or no word. The combination of source and encoder can now be regarded asaa Markov process with mn states $S_i R_j$. The transition probabilities of the combination process will be denoted by q_{ij}, and the state probabilities Q_i will satisfy

$$\sum_{j=1}^{mn} q_{ij} Q_j = Q_i \qquad i = 1, 2, \ldots mn \qquad (3.2)$$

The q_{ij} will be either zero, or equal to one of the p_{ij}, or unity. The combination process may not be ergodic, so that there may be several solutions for Q. There will also be a set of probabilities $t_{kj}(k = 0, 1)$ representing the probability that the channel word corresponding to symbol k being fed to the composite process while in state j will be generated. Again, q_{ij} can be zero, equal to one of the t_{kj}, or unity. If the channel word which is generated when k is fed to the encoder has ℓ_{kj} symbols, then the expected length of a channel word for a single state transition of the composite process is

$$L = \sum_{i=1}^{mn} Q_i \sum_{k=0}^{1} t_{kj} \ell_{kj} \qquad (3.3)$$

provided that the Q_i are uniquely determined by Eq. 3.2. The compression C is the reciprocal of L, since every state transition in the composite process corresponds to a single symbol from the source.

The method of calculation will be illustrated by encoding a "woodcut source" with the code

$$00 \rightarrow 0$$
$$01 \rightarrow 10$$
$$1 \rightarrow 11$$

A "woodcut source" will be defined as a binary source in which the probability of a 0 is $1-\beta$ if the preceding symbol was 1 and $1-\alpha$ if the preceding symbols was 0, and in which the probability of a 1 is α if the preceding symbol was 0, and in which the probability of a 1 is α if the preceding symbol was 0 and β if the preceding symbol was 0. This source, the code, and the composite process are illustrated in Fig. 3.3. If α and $1-\beta$ are small this source will generate long strings of 0's and 1's, and this is characteristic of a scanned block srawing or woodcut. If $\alpha = \beta$ this is the usual binary source with independent symbols. Outputs from either source or encoder are shown in parenthesis The stationary probabilities of the composite process must satisfy

$$\begin{bmatrix} \beta & \alpha & \alpha & \beta \\ 1-\beta & 0 & 1-\alpha & 0 \\ 0 & 1-\alpha & 0 & 1-\beta \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \end{bmatrix}$$

The unique solution, with the constraint $\sum_{i=1}^{4} Q = 1$, is :

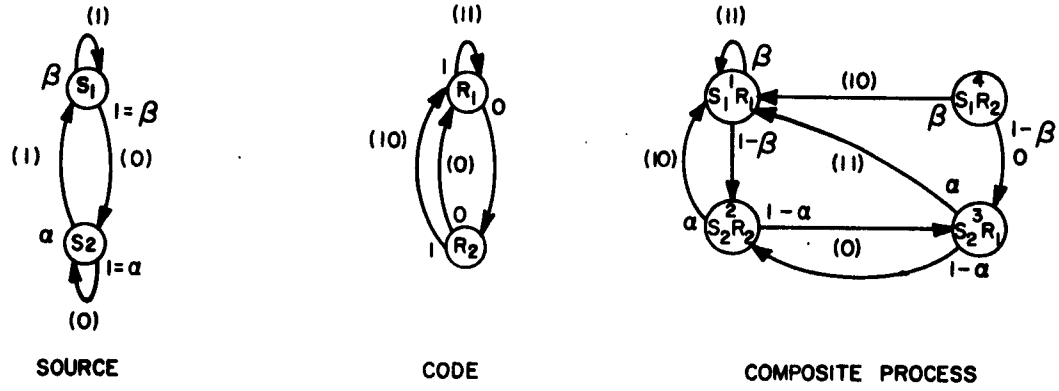

(1)
β
S1
1=β
(1)
(0)
α
S2
1=α
(0)
SOURCE
(11)
1
R1
0
(10)
(0)
0
1
R2
CODE
(11)
β
1
S1R1
(10)
4
S1R2
β
1-β
0
(10)
1-β
(11)
α
α
2
S2R2
1-α
3
S2R1
(0)
1-α
COMPOSITE PROCESS

Fig. 3.3

$$\begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \end{bmatrix} = \begin{bmatrix} \frac{\alpha}{1+\alpha-\beta} \\ \frac{1-\beta}{(2-\alpha)(1+\alpha-\beta)} \\ \frac{(1-\alpha)(1-\beta)}{(2-\alpha)(1+\alpha-\beta)} \\ 0 \end{bmatrix}$$

Note that S_1R_2 is a transient state which can never be reoccupied, and therefore $Q_4 = 0$. The average length L can now be calculated from Eq. 3.3 as

$$\frac{1}{C} = L = \frac{1 + 3\alpha - 2\alpha^2 + \alpha\beta - \beta}{(2-\alpha)(1+\alpha-\beta)} \tag{3.4}$$

A different result is obtained if the expected length of the output word is divided by the expected length of the input word:

$$L_1 = \frac{\sum_{r=1}^{w} p(r)\, y_r}{\sum_{r=1}^{w} p(r)\, x_r} \tag{3.5}$$

where p(r) is the probability of word r in the code table, y_r is the length of the r'th output word, x_r the length of the r'th input word, and w the number of words. In the present example, the probability of being in S_1 is $\frac{\alpha}{1+\alpha-\beta}$ and the probability of being in S_2 is $\frac{1-\beta}{1+\alpha-\beta}$, therefore:

$$p(1) = \underline{Pr}\ \{00\} = \frac{\alpha}{1+\alpha-\beta}(1-\alpha)(1-\beta) + \frac{1-\beta}{1+\alpha-\beta}(1-\alpha)^2$$

$$= \frac{(1-\alpha)(1-\beta)}{1+\alpha-\beta}$$

$$p(2) = \underline{Pr}\ \{01\} = \frac{\alpha(1-\beta)}{1+\alpha-\beta}$$

$$p(3) = \underline{P}r\ \{1\} = \frac{\alpha}{1 + \alpha - \beta}$$

$$L_1 = \frac{(1 + \alpha)(1 - \beta) + 2\alpha}{2(1 - \beta) + \alpha} \qquad (3.6)$$

This is not the same as L. This shows, among other things, that Huffman's method for deciding on the channel word set will not give the most efficient code even if the source word set is kept fixed. The reason is that the occurrence of various source words are not statistically independent events. Smallness of the L_1 is not an indication of the codes efficientcy unless the words occur independently.

Conclusions and Program for Next Interval

It is expected that the methods for analyzing Turing machines and computers can be used to lead to more definite conclusions as the the efficiency of realizing logical functions with sequenctial machines. The four state machines will be tabulated and classified according to their properties. An attempt will be made to derive necessary and sufficient conditions for resettability by extending the classes of non-resettable machines considered above.

REFERENCES

1. A. E. Laemmel, "Fourth Semi-Annual Report; Study on Application of Coding Theory", Report No. PIBMRI-895.4-62, Polytechnic Inst. of Brooklyn, 18 July 1962.
2. C. E. Shannon, "A Universal Turing Machine with Two Internal States", in Automata Studies", ed. by C. E. Shannon and J. McCarthy, Princeton U. Press, 1956.
3. M. Minsky, "A 6-Symbol 7-State Universal Turing Machine", M. I. T. Lincoln Laboratory Report 54 G-0027, 17 Aug. 1960.
4. E. M. Grabbe, S. Ramo, and D. E. Wooldridge, "Handbook of Automation, Computation and Control", Vol. 2, "Computers and Data Processing", John Wiley and Sons, New York, 1959.
5. "Control Data 160-A Computer, Programming Manual", Control Data Corp., Minneapolis, Minn.
6. "Honeywell 800, Programmers' Reference Manual", Minneapolis-Honeywell Datamatic Division, Wellesley Hills, Mass., 1960.
7. S. Greenwald, R. C. Haueter and S. N. Alexander, "SEAC", Proc. of IRE, Vol. 41, pp. 1300-1313, Oct. 1953.
8. M. H. Weik, "A Third Survey of Domestic Electronic Digital Computing Systems", Ballistic Research Laboratories Report No. 1115, March 1961, Aberdeen Proving Ground, Md.
9. C. E. Shannon, "The Synthesis of Two-Terminal Switching Circuits" (see Section 8), B. S. T. J. Vol. 28, pp. 59-98, Jan. 1949.
10. D. Hilbert and W. Ackermann, "Principles of Mathematical Logic", Chlesea, New York, 1950.
11. A. N. Whitehead and B. Russell, "Principia Matematica to * 56", Cambridge, 1962. (see page XVI)
12. A. E. Laemmel, "A General Class of Discrete Codes and Certain of Their Properties", Report R-459-55, Polytechnic Institute of Brooklyn, 11 Jan. 1955.
13. A. E. Laemmel, "Third Semi-Annual Report; Study on Application of Coding Theory", Report No. PIBMRI-895.3-62, 15 Jan. 1962.
14. A. E. Laemmel, "Second Semi-Annual Report; Study on Application of Coding Theory", Report No. PIBMRI-895.2-61, 31 Aug. 1961.

15. W. Feller, "An Introduction to Probability Theory and Its Applications", Vol. 1, John Wiley and Sons, New York, 1950.

16. M. P. Schützenberger, "On An Application of Semi-Group Methods to Some Problems in Coding, IRE Trans. on Information Theory, Vol. IT-2, p. 47, Sept. 1956.

17. D. A. Huffman, "A Method For the Construction of Minimum Redundancy Codes", Proc. of IRE, Vol. 40, p. 1098, Sept. 1952.

18. S. Ginsburg, "An Introduction to Mathematical Machine Theory", Addison-Wesley, Reading Mass., 1962.

DISTRIBUTION LIST

CONTRACT NO. DA-36-039 sc 78972

ORGANIZATION	COPIES
OASD (R and E) Rm 3E1065 The Pentagon Washington 25, D. C. Attn: Technical Library	1
Chief Signal Officer Department of the Army Washington 25, D. C. Attn: SIGRD	1
Commanding Officer and Director U. S. Navy Electronics Lab. San Diego 52, California	1
Commander Air Force Cambridge Research Center L. G. Hanscom Field Bedford, Massachusetts Attn: CROTR	1
U. S. Command Liaison Office U. S. Army Signal Research and Development Lab. Ft. Monmouth, New Jersey	3
Corps of Engineers Liaison Office U, S. Army Signal Research and Development Lab. Ft. Monmouth, New Jersey	4
Commanding Officer U. S. Army Signal Research and Development Lab. Ft. Monmouth, N. J. Attn: Logistics Division	3
Commanding General U, S. Army Electronic Proving Ground Ft. Huachuna, Arizona	1
Commanding Officer U. S. Army Signal Research and Development Lab. Ft. Monmouth, N. J. Attn: Technical Documents Center	1
Commanding Officer U. S. Army Signal Research and Development Lab. Ft. Monmouth, N. J. Attn: Technical Information Div.	5

ORGANIZATION	COPIES
Chief of Research and Development OCS Department of the Army Washington 25, D. C.	1
Director, U. S. Naval Research Laboratory Washington 25, D. C. Attn: Code 2027	1
Commander Wright Air Development Center Wright Patterson Air Force Base, Ohio Attn: WCOSI-3	2
Commanding Officer U. S. Army Signal Equipment Support Agency Ft. Monmouth, New Jersey Attn: SIGFM/ ES-ADJ	1
U. S. Navy Electronics Liaison Office U. S. Army Signal Research and Development Lab. Ft. Monmouth, N. J.	4
Marine Corps Liaison Office U. S. Army Signal Research and Development Lab. Ft. Monmouth, N. J.	1
Commander Rome Air Development Center Air Research and Development Command Griffiss Air Force Base, N. Y. Attn: RCSSLD	1
ASTIA (TIPDR) Arlington Hall Station Arlington 12, Virginia	10
Commanding Officer U. S. Army Signal Research and Development Lab. Ft. Monmouth, N. J. Attn: SIGRA/ SL-ADJ	1
Chief U. S, Army Security Agcy. Arlington Hall Station Arlington 12, Virginia	2

ORGANIZATION	COPIES
Deputy President U. S. Army Security Agency Board Arlington Hall Station Arlington 12, Virginia	1
Massachusetts Institute of Technology Research Laboratory of Electronics Cambridge 37, Massachusetts	1
Moore School of Electrical Engineering University of Pennsylvania Philadelphia , Pennsylvania Attn: Prof. G. W. Patterson	1
Commanding Officer U. S. Army Signal Research and Development Lab. Ft. Monmouth, N. J. Attn: SIGRA/ SL-N	5